



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/886,536	06/20/2001	Stepan Sokolov	SUN1P834/P6238	6394

22434 7590 06/21/2004
BEYER WEAVER & THOMAS LLP
P.O. BOX 778
BERKELEY, CA 94704-0778

EXAMINER

MOFIZ, APUM

ART UNIT PAPER NUMBER

2175

DATE MAILED: 06/21/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary

Application No.

09/886,536

Applicant(s)

SOKOLOV ET AL.

Examiner

Apu M Mofiz

Art Unit

2175

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 29 April 2004.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1-20 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1-20 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 20 June 2001 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
 2. ☐ Certified copies of the priority documents have been received in Application No. _____.
 3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

DIANE D. MURRAY
PRIMARY PATENT EXAMINER
TECHNOLOGY CENTER 2100

Attachment(s)

- 1) ☐ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
Paper No(s)/Mail Date _____.

- 4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____.
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: _____.

DETAILED ACTION

Response to Applicant's Remarks

1. Applicant's arguments submitted on 04/29/04 with respect to claims 1-20 have been reconsidered but are not deemed persuasive for the reasons set forth below.

Applicant argues (under REMARKS section) that Chen does not teach an internal class representation.

Examiner respectfully disagrees. Chen teaches:

"Referring to FIG. 3, a schematic block diagram depicts a class structure 300 which is derived from the data structures of a Java.TM. Virtual Machine implementation. The data structures of the object-oriented processor 202 are optimized for execution in hardware. The object-oriented processor 202 operates according to an hierarchical scheme with a class table 302 defining a first level in the hierarchy of the class structure 300." ... "The class structure 300 of the object-oriented processor 202 hierarchically begins with the class table 302. The class table 302 is a list of all possible classes used in the object-oriented processor 202." ... "The fields 306 define variables, either objects or primitives for the class." ... "The class table 302 is an array with an unlimited number of entries."

The Examiner interprets the class structure shown in the text excerpts above as the internal class representation.

Applicant argues (under REMARKS section) that Chen does not teach a reference identifier with one or more entries, such that each of the entries corresponds to a field of a Java object.

Examiner respectfully disagrees. Chen teaches:

Referring to FIG. 5, the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or class pointer classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in the field array 306 have a fixed length. For a particular entry in the field array 306, the first eight bytes store the value of the entry which is an object 502. The object 502 has one of several forms including an offset for an instance variable, a value for primitive data types, and object pointer objPtr for an array of primitives, an instance of a class or the like. If an object 502 is not a primitive, objPtr may be the address of the

Art Unit: 2175

object and objSize the length of the object. If the field contains an instance variable, the object field is the field offset of the object instantiated from the class.

The second entry in the field array 306 is a class pointer classPtr 504 entry which specifies the address of the class containing the field. The third entry in the field array 306 is a signature 506 that describes the type of field entry. The signature 506 is a combination of characters including:

```
SIGNATURE.sub.-- ARRAY '['
SIGNATURE.sub.-- BYTE 'B'
SIGNATURE.sub.-- CHAR 'C'
SIGNATURE.sub.-- FLOAT 'F'
SIGNATURE.sub.-- DOUBLE 'D'
SIGNATURE.sub.-- INT 'I'
SIGNATURE.sub.-- LONG 'J'
SIGNATURE.sub.-- CLASS 'L'
SIGNATURE.sub.-- ENDCLASS ';'
SIGNATURE.sub.-- SHORT 'S'
SIGNATURE.sub.-- VOID 'V'
SIGNATURE.sub.-- BOOLEAN 'Z'
```

For example, a label '[I' designates that a field is an integer array. A label of 'Labc' designates that a field is an object of class abc.

The fourth entry in the field array 306 is a name 508 that designates the name of the field. Name 508 is a pointer to a string in the constant pool 304. All strings in the constant pool 304 contain a 32-bit key field at the first location.

The fifth entry in the field array 306 is an access 510 element that designates the access flag of the field. Access flag bits include, as follows:

```
ACC.sub.-- PUBLIC 0x0001
ACC.sub.-- PRIVATE 0x0002
ACC.sub.-- PROTECTED 0x0004
ACC.sub.-- STATIC 0x0008
ACC.sub.-- FINAL 0x0010
ACC.sub.-- VOLATILE 0x0040
ACC.sub.-- TRANSIENT 0x0080
```

From the above text excerpts and the Figure 3, it is clear that the size of the field array (i.e. the reference identifier) is equal to the number of fields in the class. Each field (or the applicant's entry) in the array consists of a 24-byte entry. Each field in the array corresponds to a field (or attribute) of the class. 24 byte is divided into 8 bytes (502), 4 bytes (504), 4 bytes (506), 4 bytes (508) and 4 bytes (510). Each contains information

Art Unit: 2175

about the field. In Java language has two categories of data types: primitive and reference (i.e. the applicant verify this from an introductory Java language book or by going to the Sun web site.). The entries (i.e. the 24 bytes) in each of the field array entries are clearly shown to indicate the data type (i.e. if the field data is a primitive or reference) of the field. The applicant may have misinterpreted 8,4,4,4,4 bytes, which is stated as entries by Chen as entries of the field array. Each entry/field in the field array may contain more than reference information.

Claim Rejections - 35 USC § 102

2. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

3. Claims 1,2,4,7,8,18,19 and 20 are rejected under 35 U.S.C. 102(e) as being anticipated by Tao Shinn Chen (U.S. Patent No. 6,003,038 and Chen hereinafter).

As to claim 1, Chen teaches a Java computing environment (i.e. Java Virtual Machine) (Abstract), an internal class representation (i.e. *"The object-oriented processor 202 functions on an object-oriented basis of a class structure that is defined in terms of a memory layout. With the defined class structure, the object-oriented processor 202 derives a class pointer 208 to create objects as instances of classes and execute the created objects"*) (col 4, lines 35-45) suitable for use by a Java virtual machine (col 4, lines 10-20), said internal class representation (col 4, lines 35-45) comprising: a reference

identifier (i.e. field array) (Fig.5; col 7, lines 48-67) having one or more entries (i.e. one or more fields) (Fig.5; col 7, lines 48-67), wherein each of said one or more entries (i.e. each array element of the field array correspond to a field of the class) (Fig.5; col 7, lines 48-67) correspond to a field of a Java object (i.e. instance of a class) (Fig.5; col 7, lines 48-67); and wherein each of said one or more entries can be used to indicate whether a corresponding field of said Java object is a reference to another Java object (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length."* ... *"The third entry in the field array 306 is a signature 506 that describes the type of field entry"*) (From the above text excerpts and the Figure 3, it is clear that the size of the field array (i.e. the reference identifier) is equal to the number of fields in the class. Each field (or the applicant's entry) in the array consists of a 24-byte entry. Each field in the array corresponds to a field (or attribute) of the class. 24 byte is divided into 8 bytes (502), 4 bytes (504), 4 bytes (506), 4 bytes (508) and 4 bytes (510). Each contains information about the field. In Java language has two categories of data types: primitive and reference (i.e. the applicant verify this from an introductory Java language book or by going to the Sun web site.). The entries (i.e. the 24 bytes) in each of the field array entries are clearly shown to indicate the data type (i.e. if the field data is a primitive or reference) of the field. The applicant may have misinterpreted 8,4,4,4,4 bytes, which is stated as entries by Chen as entries of the field array. Each entry/field in the field array may contain more than reference information.) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3).

As to claim 2, Chen teaches that the reference identifier is an array of bytes (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte*

content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length.” ... *“The third entry in the field array 306 is a signature 506 that describes the type of field entry”*) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3); and wherein the size of said reference identifier (i.e. the field array) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3) is the same as the number of fields of said Java object (i.e. the size of the field array is the number of fields in the instantiated object i.e. fieldCount) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3).

As to claim 4, Chen teaches that in a Java computing environment (i.e. Java Virtual Machine) (Abstract), an internal class representation (i.e. *“The object-oriented processor 202 functions on an object-oriented basis of a class structure that is defined in terms of a memory layout. With the defined class structure, the object-oriented processor 202 derives a class pointer 208 to create objects as instances of classes and execute the created objects”*) (col 4, lines 35-45) suitable for use by a Java virtual machine (col 4, lines 10-20), said internal class representation (col 4, lines 35-45) comprising: a reference identifier (i.e. field array) (Fig.5; col 7, lines 48-67) having one or more entries (i.e. one or more fields) (Fig.5; col 7, lines 48-67), wherein each of said one or more entries (i.e. one or more fields) (Fig.5; col 7, lines 48-67) correspond to a field of a Java object (i.e. instance of a class) (Fig.5; col 7, lines 48-67); and wherein each of said one or more entries (Fig.5; col 7, lines 48-67) can be used to indicate whether corresponding fields of said Java object is a reference to another Java object (i.e. *“the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length.*” ... *“The third entry in the field array 306 is a signature 506 that describes the type of field entry”*) (From the above text excerpts and the Figure 3, it is clear that the size of the field array (i.e. the reference

identifier) is equal to the number of fields in the class. Each field (or the applicant's entry) in the array consists of a 24-byte entry. Each field in the array corresponds to a field (or attribute) of the class. 24 byte is divided into 8 bytes (502), 4 bytes (504), 4 bytes (506), 4 bytes (508) and 4 bytes (510). Each contains information about the field. In Java language has two categories of data types: primitive and reference (i.e. the applicant verify this from an introductory Java language book or by going to the Sun web site.). The entries (i.e. the 24 bytes) in each of the field array entries are clearly shown to indicate the data type (i.e. if the field data is a primitive or reference) of the field. The applicant may have misinterpreted 8,4,4,4,4 bytes, which is stated as entries by Chen as entries of the field array. Each entry/field in the field array may contain more than data type reference/primitive information.)

(Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3); wherein said reference identifier (i.e. field array)

(Fig.5; col 7, lines 48-67) is an array of bytes (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length."* ... *"The third entry in the field array 306 is a signature 506 that describes the type of field entry"*) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3), where

the size of said array of bytes is the same as the number of fields of said Java object

(Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3), and wherein an entry of said array of bytes is

set to a predetermined non-zero value (i.e. *"For a particular entry in the field array 306, the first eight bytes store the value of the entry which is an object 502. The object 502 has one of several forms including an offset for an instance variable, a value for primitive data types, and object pointer objPtr for an array of primitives, an instance of a class or the like. If an object 502 is not a primitive, objPtr may be the address of the object and objSize the length of the object. If the field contains an instance variable, the object field is the field offset of the object instantiated from the class."*). The previous text excerpts clearly indicate that the array of bytes in the field array is set to some non-zero value whether the field is of a primitive data type or of reference type.) (Abstract; Fig.

3; Fig.5; col 7, lines 48-67; col 8, lines 1-3) to indicate that the corresponding field of said Java object is not a reference to another object (Abstract; Fig. 3; Fig.5; col 7, lines 48-67; col 8, lines 1-3).

As to claim 7, Chen teaches wherein said array of bytes (i.e. the field array) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3) is allocated and set to appropriate values during load time (i.e. *"The Java Virtual Machine has a method area that is shared among all threads. The method area is analogous to the storage area for compiled code of a conventional language. The method area stores pre-class structures including the constant pool, field and method data, and the code for methods and constructors. The method area is created on virtual machine start-up."*) (col 10, lines 1-20).

As to claim 8, Chen teaches that the reference identifier (i.e. the field array) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3) is allocated during load time (i.e. *"The Java Virtual Machine has a method area that is shared among all threads. The method area is analogous to the storage area for compiled code of a conventional language. The method area stores pre-class structures including the constant pool, field and method data, and the code for methods and constructors. The method area is created on virtual machine start-up."*) (col 10, lines 1-20).

As to claim 18, Chen teaches a computer readable media including computer program code for an internal class representation (i.e. *"The object-oriented processor 202 functions on an object-oriented basis of a class structure that is defined in terms of a memory layout. With the defined class structure, the object-oriented processor 202 derives a class pointer 208 to create objects as instances of classes and execute the created objects"*) (col 4, lines 35-45) suitable for use by a Java virtual machine, said computer readable media comprising: computer program code for a reference identifier

(i.e. field array) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3) having one or more entries (i.e. one or more fields) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3), wherein each of said one or more entries correspond to a field of a Java object (i.e. each array element of the field array correspond to a field of the class) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3); and wherein each of said one or more entries can be used to indicate whether corresponding fields of said Java object is a reference to another Java object (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length."* ... *"The third entry in the field array 306 is a signature 506 that describes the type of field entry"*) (From the above text excerpts and the Figure 3, it is clear that the size of the field array (i.e. the reference identifier) is equal to the number of fields in the class. Each field (or the applicant's entry) in the array consists of a 24-byte entry. Each field in the array corresponds to a field (or attribute) of the class. 24 byte is divided into 8 bytes (502), 4 bytes (504), 4 bytes (506), 4 bytes (508) and 4 bytes (510). Each contains information about the field. In Java language has two categories of data types: primitive and reference (i.e. the applicant verify this from an introductory Java language book or by going to the Sun web site.). The entries (i.e. the 24 bytes) in each of the field array entries are clearly shown to indicate the data type (i.e. if the field data is a primitive or reference) of the field. The applicant may have misinterpreted 8,4,4,4,4 bytes, which is stated as entries by Chen as entries of the field array. Each entry/field in the field array may contain more than data type reference/primitive information.) (Abstract; Fig. 3; Fig.5; col 7, lines 48-67; col 8, lines 1-3).

As to claim 19, Chen teaches that the reference identifier is an array of bytes (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte*

Art Unit: 2175

content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length." ... "The third entry in the field array 306 is a signature 506 that describes the type of field entry" (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3); and wherein the size of said reference identifier (i.e. the field array) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3) is the same as the number of fields of said Java object (i.e. the size of the field array is the number of fields in the instantiated object i.e. fieldCount) (Abstract; Fig. 2; Fig.5; col 7, lines 48-67; col 8, lines 1-3).

As to claim 20, Chen teaches that the array of bytes (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length." ... "The third entry in the field array 306 is a signature 506 that describes the type of field entry"*) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3) is allocated and set to appropriate values during load time (i.e. *"The Java Virtual Machine has a method area that is shared among all threads. The method area is analogous to the storage area for compiled code of a conventional language. The method area stores pre-class structures including the constant pool, field and method data, and the code for methods and constructors. The method area is created on virtual machine start-up."*) (col 10, lines 1-20).

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and

Art Unit: 2175

the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

5. Claims 3, 5,6, and 9-17 are rejected under 35 U.S.C. 103(a) as being unpatentable over Tao Shinn Chen (U.S. Patent No. 6,003,038 and Chen hereinafter) in view of Adl-Tabatabai et al. (U.S. Patent No. 6,093,216 and Adl-Tabatabai hereinafter)).

The teachings of Chen have been discussed above.

As to claim 3, Chen teaches an array of bytes (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length."* ... *"The third entry in the field array 306 is a signature 506 that describes the type of field entry"*) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3).

Chen does not explicitly teach setting an entry to zero to indicate that the corresponding field of said Java object is not a reference to another Java object.

Adl-Tabatabai teaches setting an entry to zero to indicate that the corresponding field of said Java object is not a reference to another Java object (i.e. *"Specifically, when a variable is assigned a reference value, the corresponding bit in the bit vector is set. When a variable is assigned a non-reference value, the corresponding bit in the bit vector is cleared."*) (Abstract).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Chen with the teachings of Adl-Tabatabai to include setting an entry to zero to indicate that the corresponding field of said Java object is not a reference to another Java object with the motivation to

Art Unit: 2175

determine which variables contain references to objects at garbage collection sites (Adl-Tabatabai , col3, lines 34-36) and the Garbage Collector to know which variable contain reference values and easily find the root set (Adl-Tabatabai , col 7, lines 35-37) and to improve performance (Adl-Tabatabai, col 1, lines 60-62).

As to claim 5, Chen does not explicitly teach that the predetermined non-zero value is equal to 1.

Adl-Tabatabai teaches that the predetermined non-zero value is equal to 1 (i.e. *"Specifically, when a variable is assigned a reference value, the corresponding bit in the bit vector is set. When a variable is assigned a non-reference value, the corresponding bit in the bit vector is cleared."*) (The Examiner asserts that the predetermined value is 1 or 0, i.e. set or cleared) (Abstract).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Chen with the teachings of Adl-Tabatabai to include that the predetermined non-zero value is equal to 1 with the motivation to determine which variables contain references to objects at garbage collection sites (Adl-Tabatabai , col3, lines 34-36) and the Garbage Collector to know which variable contain reference values and easily find the root set (Adl-Tabatabai , col 7, lines 35-37) and to improve performance (Adl-Tabatabai, col 1, lines 60-62).

As to claim 6, Chen does not explicitly teach that an entry of said array of bytes is set to zero to indicate that the corresponding field of said Java object is not a reference to another Java object; and wherein an entry of said array of bytes is set to a

Art Unit: 2175

predetermined nonzero value to indicate that the corresponding field of said Java object is not a reference to another Java object.

Adl-Tabatabai teaches that an entry of said array of bytes is set to zero to indicate that the corresponding field of said Java object is not a reference to another Java object; and wherein an entry of said array of bytes is set to a predetermined nonzero value to indicate that the corresponding field of said Java object is not a reference to another Java object (i.e. *"Specifically, when a variable is assigned a reference value, the corresponding bit in the bit vector is set. When a variable is assigned a non-reference value, the corresponding bit in the bit vector is cleared."*) (The Examiner asserts that the predetermined value is 1 or 0, i.e. set or cleared) (Abstract).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Chen with the teachings of Adl-Tabatabai to include that an entry of said array of bytes is set to zero to indicate that the corresponding field of said Java object is not a reference to another Java object; and wherein an entry of said array of bytes is set to a predetermined nonzero value to indicate that the corresponding field of said Java object is not a reference to another Java object with the motivation to determine which variables contain references to objects at garbage collection sites (Adl-Tabatabai, col3, lines 34-36) and the Garbage Collector to know which variable contain reference values and easily find the root set (Adl-Tabatabai, col 7, lines 35-37) and to improve performance (Adl-Tabatabai, col 1, lines 60-62).

As to claim 9, Chen teaches a method for generating a reference identifier (i.e. field array) (col 7, lines 48-67; col 8, lines 1-3) for a Java object (i.e. each array element of the field array correspond to a field of the class/ instantiated class or object) (col 7, lines 48-67; col 8, lines 1-3), said method comprising: allocating (i.e. *"The Java Virtual Machine has a method area that is shared among all threads. The method area is analogous to the storage area for compiled code of a conventional language. The method area stores pre-class structures including the constant pool, field and method data, and the code for methods and constructors. The method area is created on virtual machine start-up."*) (col 10, lines 1-20) a reference identifier (i.e. field array) (col 7, lines 48-67; col 8, lines 1-3) for said Java object (i.e. instance of a class) (col 7, lines 48-67; col 8, lines 1-3); and wherein said reference identifier indicates which fields of said Java object are references (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length." ... "The third entry in the field array 306 is a signature 506 that describes the type of field entry"*) (The Examiner asserts that the third entry in the field array describes the type of the field entry. A variable of array type holds a reference to an object. If all of the fields in the class is of type array then the object-oriented processor/garbage collector can easily determine if the corresponding field is a reference) (col 7, lines 48-67; col 8, lines 1-3).

Chen does not explicitly teach reading a class file associated with a Java object; identifying fields of said Java object that are references.

Adl-Tabatabai teaches reading a class file associated with a Java object (i.e. *"In step 505, the compiler first scans the Java class file. Then at step 510, the JIT compiler allocates memory space for the program. The memory space is used for program execution and for storing objects declared in the code. Objects may be assigned space on heap, runtime stack, cache, or physical machine registers. The JIT compiler knows when*

objects are assigned and become references.") (Abstract; col 5, lines 50-67); identifying fields of said Java object that are references (i.e. *"The present invention introduces a method for run-time tracking of object references in computer code and determining which variables contain references to objects at garbage collection sites. The method of the present invention first creates a bit vector in memory. Each bit is assigned a unique bit within this bit vector."*) (Abstract).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Chen with the teachings of Adl-Tabatabai to include reading a class file associated with a Java object; identifying fields of said Java object that are references with the motivation to determine which variables contain references to objects at garbage collection sites (Adl-Tabatabai, col 3, lines 34-36) and the Garbage Collector to know which variable contain reference values and easily find the root set (Adl-Tabatabai, col 7, lines 35-37) and to improve performance (Adl-Tabatabai, col 1, lines 60-62).

As to claim 10, Chen teaches that the said method is performed at load time by a virtual machine (i.e. *"The Java Virtual Machine has a method area that is shared among all threads. The method area is analogous to the storage area for compiled code of a conventional language. The method area stores pre-class structures including the constant pool, field and method data, and the code for methods and constructors. The method area is created on virtual machine start-up."*) (col 10, lines 1-20).

As to claim 11, Chen teaches that the reference identifier (i.e. the field array) (col 7, lines 48-67; col 8, lines 1-3) is an array of bytes (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or*

Art Unit: 2175

classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length.” ... *“The third entry in the field array 306 is a signature 506 that describes the type of field entry”*) (col 7, lines 48-67; col 8, lines 1-3); and wherein the size of said reference identifier is the same as the number of fields of said Java object (i.e. the size of the field array is the number of fields in the instantiated object i.e. fieldCount) (col 7, lines 48-67; col 8, lines 1-3).

As to claim 12, Chen teaches an array of bytes (i.e. *“the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length.”* ... *“The third entry in the field array 306 is a signature 506 that describes the type of field entry”*) (col 7, lines 48-67; col 8, lines 1-3).

Chen does not explicitly teach setting an entry to zero to indicate that the corresponding field of said Java object is not a reference to another Java object.

Adl-Tabatabai teaches setting an entry to zero to indicate that the corresponding field of said Java object is not a reference to another Java object (i.e. *“Specifically, when a variable is assigned a reference value, the corresponding bit in the bit vector is set. When a variable is assigned a non-reference value, the corresponding bit in the bit vector is cleared.”*) (Abstract).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Chen with the teachings of Adl-Tabatabai to include setting an entry to zero to indicate that the corresponding field of said Java object is not a reference to another Java object with the motivation to determine which variables contain references to objects at garbage collection sites (Adl-

Tabatabai , col 3, lines 34-36) and the Garbage Collector to know which variable contain reference values and easily find the root set (Adl-Tabatabai , col 7, lines 35-37) and to improve performance (Adl-Tabatabai, col 1, lines 60-62).

As to claim 13, Chen teaches wherein an entry of said array of bytes is set to a predetermined non-zero value (i.e. *"For a particular entry in the field array 306, the first eight bytes store the value of the entry which is an object 502. The object 502 has one of several forms including an offset for an instance variable, a value for primitive data types, and object pointer objPtr for an array of primitives, an instance of a class or the like. If an object 502 is not a primitive, objPtr may be the address of the object and objSize the length of the object. If the field contains an instance variable, the object field is the field offset of the object instantiated from the class."*). The previous text excerpts clearly indicate that the array of bytes in the field array is set to some non-zero value whether the field is of a primitive data type or of reference type to indicate their types.) (Abstract; Fig. 3; Fig.5; col 7, lines 48-67; col 8, lines 1-3) to indicate that the corresponding field of said Java object is not a reference to another object (Abstract; Fig. 3; Fig.5; col 7, lines 48-67; col 8, lines 1-3).

As to claim 14, Chen teaches a method for determining whether a field of a Java object is a reference to another Java object (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length."* ... *"The third entry in the field array 306 is a signature 506 that describes the type of field entry"*) (The Examiner asserts that the third entry in the field array describes the type of the field entry. A variable of array type holds a reference to an object. If all of the fields

in the class is of type array then the object-oriented processor/garbage collector can easily determine if the corresponding field is a reference) (Abstract; Fig.5; col 7, lines 48-67; col 8, lines 1-3).

Chen does not explicitly teach identifying the internal class representation for the Java object; identifying a reference identifier in said internal class representation; reading a portion of said reference identifier that represents said field of said Java object; and determining whether the value stored in said portion of said reference identifier is equal to a predetermined value.

Adl-Tabatabai teaches identifying the internal class representation (i.e. *"After the class file 390 has been downloaded, it is passed into the Java Virtual Machine 330, which then verifies the downloaded class file at step 215." ... "After the Java program has been verified, a Java "Just-In-Time" (JIT) compiler 333 compiles the Java class file and generates compiled Java code 340 in the form of native processor code at step 260." ... "During program execution, numerous objects or variables may be declared and used within the program. An object, also referred to as a cell or node, is a run-time notion; any object is an instance of a certain class, created at execution time and made of a number of fields."*) (The Examiner asserts that the virtual machine loads the class file and internally represents the information as an internal class representation.) (col 4, lines 1-67) for the Java object (i.e. the instance of a class) (col 4, lines 1-67); identifying a reference identifier (i.e. the bit vector) (Abstract) in said internal class representation (col 4, lines 1-67); reading a portion (i.e. *"The tags in the bit vector are used at garbage collection time to decide which variables actually hold valid references. The garbage collector (GC) needs to discover which variables contain reference values and compute the root set."*) (col 6, lines 54-67) of said reference identifier (i.e. bit vector) (col 6, lines 54-67) that represents said field of said Java object (i.e. instance of a class) (Abstract); and determining whether the value stored in said portion (i.e. *"The tags in the bit vector are used at garbage collection time to decide which variables actually hold valid references. The garbage collector (GC) needs to discover which variables contain reference values and compute the root set."*) (col 6, lines 54-67) of said reference identifier

Art Unit: 2175

is equal to a predetermined value (i.e. *"Specifically, when a variable is assigned a reference value, the corresponding bit in the bit vector is set. When a variable is assigned a non-reference value, the corresponding bit in the bit vector is cleared."*) (The Examiner asserts that the predetermined value is 1 or 0, i.e. set or cleared) (Abstract).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Chen with the teachings of Adl-Tabatabai to include identifying the internal class representation for the Java object; identifying a reference identifier in said internal class representation; reading a portion of said reference identifier that represents said field of said Java object; and determining whether the value stored in said portion of said reference identifier is equal to a predetermined value with the motivation to determine which variables contain references to objects at garbage collection sites (Adl-Tabatabai, col3, lines 34-36) and the Garbage Collector to know which variable contain reference values and easily find the root set (Adl-Tabatabai, col 7, lines 35-37) and to improve performance (Adl-Tabatabai, col 1, lines 60-62).

As to claim 15, Chen teaches that the said method is performed by a Java virtual machine at runtime (i.e. *"Hardware processor 100 provides similar advantages for other virtual machine stack-based architectures as well as for virtual machines utilizing features such as garbage collection and thread synchronization"* ... *"Runtime data areas include a program counter (pc) register, a Java stack, a heap, a method area, a constant pool, and native method stacks"*) (Examiner asserts that Java Virtual Machine needs to resolve object references at run time e.g. during garbage collection) (col 3, lines 35-67; col 9, lines 55-67).

As to claim 16, Chen teaches that the reference identifier (i.e. the field array) is an array of bytes (i.e. *"the field array 306 is a collection of the fields defined in the current class. The field array 306 is used to store a static variable or an instance variable. A variable can be either an array or an object. The field structure of the field array 306 has four entries including a four-byte class index or classPtr 504, a string signature 506, an eight-byte content element holding the object 502, and a two-byte access 510 element. The entries in field array 306 have a fixed length."* ... *"The third entry in the field array 306 is a signature 506 that describes the type of field entry"*) (col 7, lines 48-67; col 8, lines 1-3); and wherein the size of said reference identifier is the same as the number of fields of said Java object (i.e. the size of the field array is the number of fields in the instantiated object i.e. fieldCount) (Fig. 2; Fig. 5; col 7, lines 48-67; col 8, lines 1-3).

As to claim 17, Chen does not explicitly teach that the predetermined value can be 1 or zero.

Adl-Tabatabai teaches that the predetermined value can be 1 or zero (i.e. *"Specifically, when a variable is assigned a reference value, the corresponding bit in the bit vector is set. When a variable is assigned a non-reference value, the corresponding bit in the bit vector is cleared."*) (The Examiner asserts that the predetermined value is 1 or 0, i.e. set or cleared) (Abstract).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Chen with the teachings of Adl-Tabatabai to include that the predetermined value can be 1 or zero with the motivation to determine which variables contain references to objects at garbage collection sites (Adl-Tabatabai, col3, lines 34-36) and the Garbage Collector to know which variable contain reference values and easily find the root set (Adl-Tabatabai, col 7, lines 35-37) and to improve performance (Adl-Tabatabai, col 1, lines 60-62).

Points of Contact

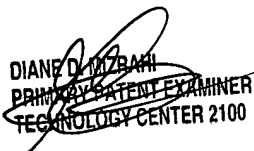
6. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Apu M. Mofiz whose telephone number is (703) 605-4240. The examiner can normally be reached on Monday – Thursday 8:00 A.M. to 4:30 P.M.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Dov Popovici can be reached at (703) 305-3830. The fax numbers for the group is (703) 872-9306.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 305-9600.

Apu M. Mofiz
Patent Examiner
Technology Center 2100

June 16,2004


DIANE D. M. Z. B. H.
PRIMARY PATENT EXAMINER
TECHNOLOGY CENTER 2100